Submission in Response to NSF CI 2030 Request for Information

DATE AND TIME: 2017-02-03 10:21:56
REFERENCE NO: 175

Author Names & Affiliations

- Leigh Orf - University of Wisconsin

Contact Email Address (for NSF use only)

(Hidden)

Research Domain, discipline, and sub-discipline

Numerical modeling of thunderstorms, meteorology, mesoscale meteorology

Title of Submission

Big data, efficient computing, and fault tolerance

Abstract (maximum ~200 words).

As someone who has been using supercomputing hardware for two decades, and who is currently doing simulation work on Blue Waters, I have experienced the transition from monlithic shared-memory hardware to massively parallel supercomputers with sophisticated shared-memory nodes. To someone from 40 years ago, today's supercomputers would look like "a supercomputer of supercomputers."

From my perspective, the biggest challenges can be summarized in three categories:

(a) DATA: Dealing with I/O inefficiencies, and managing data organization/archiving (A piece of the "big data" puzzle)

(b) EFFICIENT COMPUTING: There is a growing mismatch between the hardware topology and the software run on these machines (many legacy codes are MPI only, or - gasp - serial, and scientists are not necessarily interested or equipped to retrofit their codes to take advantage of these topologies and/or accelerators)

(c) FAULT TOLERANCE: With millions of cores, some will crash; should faults be handled by hardware or software?

Question 1 Research Challenge(s) (maximum ~1200 words): Describe current or emerging science or engineering research challenge(s), providing context in terms of recent research activities and standing questions in the field.

I will speak primarily of challenges in my own field. Atmospheric simulations are conducted on scales ranging from planetary scale (covering the globe) to microscale (covering hundreds of square meters). In atmospheric science, operational forecasts (forecasts of the current weather provided to the general population) need to be executed in a timely manner in order to be useful, whereas basic research simulations do not have this restriction. Atmospheric models such as WRF tend to scale sublinearly as you get out to many (tens of thousands of) cores. Memory bandwidth seems to be a main factor in our ability to scale well, as does frequent communication between processing cores/nodes. A relatively new approach towards improving operational weather forecasts involves ensemble forecasts, where dozens of simulations, each with slightly different parameters or initial conditions, are run concurrently, and statistical information is calculated in order to provide a probabilistic forecast. Ensemble simulations are the future, as they provide much more robust information about the atmosphere's future state than a single deterministic forecast. However, the requirement for dozens of concurrent simulations requires significant supercomputing resources as well as code that can efficiently utilize the hardware. Much post-processing is typically done following model integration, requiring a fast I/O subsystem. My main concern is that models such as WRF will "plateau out" to the point where adding more computing resources results in little, no, or negative performance increases. Hence, it may be necessary to create a new modeling system from the ground up if WRF (and other commonly used operational models) cannot be modified to take advantage of the new supercomputing platforms.-

The research side of things has similar challenges. Research simulations will tend to be larger than operational simulations since they do not contain a time constraint to completion. Many of the issues that operational models will face will first be faced by the research community (and hence the "fixes" can be passed down to the operational world). In order to move the field forward, research models will need to efficiently utilize the next generation supercomputers. It is difficult to know the exact challenges we will face before the hardware has been determined, but I am assuming many fat nodes, each containing thousands of cores and accelerators on the board. It does not make any sense to use MPI on each node, a common practice today. We will need to think very carefully about how to first best utilize one of these fat nodes, and then use MPI to exchange data between them. I suspect this will require a much more sophisticated approach towards OpenMP programming than typically occurs in today's atmospheric models. Only by utilizing the latest hardware efficiently can we make new breakthrough discoveries in the research arena, and provide better probabilistic forecasts operationally.

Finally, data storage and archiving is very important in order to be able to reproduce prior work as well as harvest the huge amount of data produced on these machines past the date of their decommissioning. It would be a shame to have to throw away lots of good data simply because we don't have a place to put it.

Question 2 Cyberinfrastructure Needed to Address the Research Challenge(s) (maximum ~1200 words): Describe any limitations or absence of existing cyberinfrastructure, and/or specific technical advancements in cyberinfrastructure (e.g. advanced computing, data infrastructure, software infrastructure, applications, networking, cybersecurity), that must be addressed to accomplish the identified research challenge(s).

DATA: We need asynchronous I/O - yesterday!

In the "old days" on single shared node supercomputers, saving data at a given stage in a simulation involved writing a single file to disk. This single file was written from a block of shared memory on the supercomputer. All integration would halt until the file write operations were completed.

When distributed memory architectures ("Beowulf clusters") became more common, scientists had to choose between saving a single file as before (using MPI to communicate data to the write node), or saving a file for each shared-memory node, either to local storage or to a shared file system. In the former case, the single file was written in stages as the write node received data from every compute node and did the work of making sure the written file was organized as if it had been written by a single node, and all integration would halt during the entire process of collecting data and writing it. In the latter case, all integration would halt until each and every file written by each compute node was complete. In most cases, the "one file per node" paradigm was the fastest; however, this approach fragments the model domain and requires some care to read back for visualization and analysis.

Fast forwarding to today, there have been improvements made both in hardware and software for handling I/O. Parallel filesystems (i.e., Lustre, gpfs) and parallel file structures (i.e., HDF5, netCDF4) have made it possible to write single files from distributed memory architectures without the domain scientist having to worry about building the file. However, as the number of compute cores has increased into the hundreds of thousands, writing a single file using parallel I/O becomes inefficient and unweildy (who wants to analyze, move, or archive a 20 TB file?). Hence, distributing model output amongst multiple files, even if parallel I/O is utilized, remains the most efficient way to get data from memory to disk.

Some novel approaches (e.g., Damaris) involve utilizing "dedicated I/O" wherein a single core on a shared memory node is dedicated entirely to I/O, and the other cores on the node do computing but no I/O. The advantage of this approach is that you can overlap I/O with computation. However, Damaris is an entire set of code that must be merged with and compiled along side the simulation code, and this can be a barrier for adoption.

Another modern approach that I have used with good results is buffering up to hundreds of write cycles to memory before sending data to disk. In this approach, many time steps' worth of data are written to a file that grows in memory (in my case, utilizing the HDF5 core driver), without touching the I/O subsystem. Because memory bandwidth is orders of magnitude greater than I/O bandwidth (and latencies associated with writing many files to disk are much greater than writing files to memory), each time slice written to memory occurs much faster than it would have had each slice been written to files on disk. And, when the time for writing data arrives due to memory exhaustion or choice, many large files are sent to disk concurrently, which is a paradigm that I have found comes closest to saturating the I/O pipeline, reducing total I/O time dramatically over other methods.

What would dramatically alleviate the I/O bottleneck in many cases would be hardware-enabled asynchronous I/O. Asynchronous I/O is analogous to non-blocking MPI: Once you call the routine to send the data (i.e., write to to disk), it immedately returns so that the model can keep integrating while data finds its way to disk (similar to Damaris), rather than blocking everything until the file is finished writing. Asynchronous I/O for non-parallel writes should be much easier to implement than parallel asyncronous I/O, but the "holy grail" would be parallel asynchronous I/O.

While software solutions like Damaris and ADIOS can be utilized to improve throughput and efficiency, such functionality would best reside within hardware. Having a separate "subsystem" to do nothing but handle asyncrhonous I/O would remove the largest barrier from computing efficiency for I/O-intensive simulations. One could envision a system where data tagged for writing would be copied in memory, passed through a separate network indepenent of the communication network handling data exchange between nodes, and organized and written based upon a set of parameters chosen by the domain scientist. Having this functionality available to all domain scientists by default (without considerably changing any code) would eliminate or drastically reduce I/O wallclock time as a percentage of model execution wallclock time.

Finally, I am really hoping tomorrow's exascale machines contain no spinning platter hard drives but have all solid state drives for scratch/project space.

EFFICIENT COMPUTING: Many codes (especially legacy codes that have been maintained for many years) made the transition from shared memory to distributed memory by utilizing the Message Passing Interface (MPI) library. The earliest distributed memory supercomputers had only one CPU per node, and MPI was the best way to exchange information between nodes during execution. As time went on, multicore nodes presented a situation where 1 MPI rank per core wasn't necessarily the most efficient way to split the problem up. While modern MPI libraries are quite efficient when data is exchanged on a shared-memory node (i.e., data does not hit the network), it is not the most ideal approach; it doesn't make much sense to be using a library designed for moving data between distributed memory nodes on a shared memory node.

However, "all MPI" models persist to this day on supercomputers, and, while "decent enough" performance can be found in some models using this approach, a better approach is needed, especially for tomorrow's "manycore" architectures where each shared memory node will contain thousands of compute cores.

OpenMP is one obvious way to deal with this, but is a completely different programming model from MPI that seems to be getting adopted pretty slowly. OpenMP is designed specifically for parallelizing work on a shared memory node. OpenMP opens doors for more efficient programming strategies than "MPI everywhere". Ideally, a single MPI rank should be assigned to each shared-memory node, with OpenMP handling intranode parallelization. However, in order for this approach to be efficient, significant code rewrites are often required.

There needs to be a great push for better programming practices on tomorrow's machines, but there also needs to be a lot of input from domain scientists to hardware manufacturers. We do not need a situation where the hardware manufacturers create hardware that few can use efficiently due to the huge learning curve required to properly utilize it. The vast majority of supercomputer users are not computer scientists, hence having a hardware topology that can be exploited effectively by the compiler, or hinted at by other software, is probably going to be a big part of the next machines.

Related to this is the transition towards hybrid hardware paradigms where CPUS and GPUs (or similar) are available to programmers. Current implementations involving GPUs are still difficult to use, partly due to the way the hardware is organized. I envision future hardware paradigms including CPUs and GPUs on the same board/chip, all having the same memory address space. This would go a long ways towards more efficient computing, assuming no major code rewrites are required to utilize this hybrid architecture.

FAULT TOLERANCE: Semi-regular hardware failures are a fact of life on machines containing tens of thousands of nodes / hard drives. It is very difficult to code a model that can handle faults. Tomorrow's machines will, hopefully, handle such hardware failures without the need for code rewrites. I really don't see a way around this. Currently, a bad node just kills your job, so you have to restart from checkpoint files. On many million core machines the MTBF across the machine will likely be short and without built in fault tolerance, simulations will be regularly crashing due to hardware failures. I would hope that hardware manufacturers would be motivated to come up with a system that worked around these failures seamlessly for users of the system.

Question 3 Other considerations (maximum ~1200 words, optional): Any other relevant aspects, such as organization, process, learning and workforce development, access, and sustainability, that need to be addressed; or any other issues that NSF should consider.

1. One frustrating aspect of getting access to federal supercomputing hardware such as the NSF-sponsored Blue Waters is that you need to write a full NSF proposal to get access to the machine, but also must have funded proposal to pay those who use the machine to do science. I would very much like to see a funding paradigm wherein one proposal is written (perhaps longer than the standard 15 pages for the project description) that addresses everything, including access to the supercomputing hardware. This would also ensure that funding for machine time and salary/etc. would overlap perfectly.

2. Supercomputers such as Blue Waters have a storage system attached to the machine to archive large amounts of data. Where does this data go once a scientist's allocation ends? One aspect of modern supercomputing is that a single handful of simulations can produce enough data to study for many years beyond the typical two- to three-year allocation. Some sort of curated "data warehouse" with a front-end that allows access/visualization of data from machines like Blue Waters would ensure that good data isn't discarded. Serious consideration should be given for what happens to data after an allocation ends; while NSF will certainly want to make sure access is provided to as many scientists as possible, with machines like Blue Waters it's easy to create enough data to keep you busy for many years beyond a typicall 2 or 3 year machine allocation.

3. In the early days of Blue Waters (before it was built, even) many workshops were hosted by NCSA to aid PRAC recipients on how to best utilize the hardware. These workshops could therefore guide domain scientists in modifying existing code to run more efficiently on Blue Waters. I suspect that even more effort will be required in this vein for future architectures. It is a tough situation; scientists want to do science, but manycore exascale machines will require a significantly different programming paradigm that may not be conducive to retrofitting legacy codes. There needs to be a balance between code efficiency and the value of these legacy codes that have decades of people-years invested in them. I don't know what the solution to this problem is. I suspect some legacy codes will need a substantial rewrite to work on future machines. Who will do this, and who will pay for it? Will legacy codes be shut out from future exacscale systems completely due to their inefficiency? Now may be the time to begin addressing some of these issues.